On foundational aspects of RDF and SPARQL

Dominique Duval, Rachid Echahed, Frederic Prost

On foundational aspects of RDF and SPARQL (Revised Version)

Dominique Duval, Rachid Echahed and Frédéric Prost

CNRS and Univ. Grenoble Alpes, Grenoble, France

Abstract. We consider the recommendations of the World Wide Web Consortium (W3C) about RDF framework and its associated query language SPARQL. We propose a new formal framework based on category theory which provides clear and concise formal definitions of the main basic features of RDF and SPARQL. We define RDF graphs as well as SPARQL basic graph patterns as objects of some nested categories. This allows one to clarify, in particular, the role of blank nodes. Furthermore, we consider basic SPARQL CONSTRUCT and SELECT queries and formalize their operational semantics following a novel algebraic graph transformation approach called POIM.

1 Introduction

Graph databases are becoming a very influential technology in our society. Mastering the languages involved in the encoding of data or the formulation of queries is a necessity to elaborate robust data management systems.

In this paper, we consider the most recent recommendations of the World Wide Web Consortium (W3C) about the Resource Description Framework (RDF) [17] and the associated query language SPARQL [16] and propose a mathematical semantics of a kernel of these formalisms.

The key data structure in RDF is the structure of RDF graph. In [17, Section 3], an RDF graph is defined as a set of RDF triples, where an RDF triple has the form $(subject, predicate, object)$. The subject is either an IRI (Internationalized Resource Identifier) or a blank node, the predicate is an IRI and the object is either an IRI, a literal (denoting a value such as a string, a number or a date) or a blank node. Blank nodes are arbitrary elements as long as they differ from IRIs and literals and they do not have any internal structure: they are used for indicating the existence of a thing and the blank node identifiers are locally scoped. For instance, the triples $(Paul, knows, blank1)$ and $(blank2, knows, Henry)$ mean, respectively, that Paul knows someone and someone knows Henry. Surprisingly, a triple such as $(Paul, blank3, Henry)$ standing for "there is some relationship between Paul and Henry" is not allowed in RDF, but only in generalized RDF [17, Section 7]. Following the theoretical point of view we propose in this paper, there is no harm to consider blank predicates within RDF triples. We thus consider *data graphs* in a more general setting including RDF graphs.

The query language SPARQL for RDF databases is based on basic graph patterns, which are kinds of RDF graphs with variables [16, Section 2]. In this

paper, we consider *query graphs* which generalize basic graph patterns by allowing blanks to be predicates. The SPARQL query processor searches for triples within a given RDF database which match the triple patterns in the given basic graph pattern, and returns a multiset of solutions or an RDF graph. Considering basic graph patterns, one may wonder what is the difference between variables and blank nodes. SPARQL specifications in [16, Section 4.1.4] suggest similarities between them, whereas the opposite is made in [16, Section 16.2]. In the formalization of SPARQL we propose, blank nodes and variables are clearly distinguished by their respective roles in the definition of morphisms.

In the SPARQL recommendation [16], the SELECT query form is described lengthily. This query form can be compared to the SELECT query form of SQL, which returns a multiset of solutions. In contrast, the CONSTRUCT query form returns an RDF graph. The latter is described very shortly in [16, Section 16.2]. Following our formalization, the CONSTRUCT query form is more fundamental than the SELECT query form. Actually, we start by proposing an operational semantics for CONSTRUCT queries based on a new approach of algebraic graph transformations which we call POIM and we show afterward how SELECT queries can be easily encoded as CONSTRUCT queries.

The paper is organized as follows. Section 2 defines the objects and the morphisms of the categories of data graphs and query graphs. Section 3 introduces the POIM algebraic transformation: a rewrite rule is a cospan $L \to K \leftarrow R$ where L, K and R are basic graph patterns, and a rewrite step is made of a pushout followed by an image factorization. Afterward, in Section 4 we define two different operational semantics for CONSTRUCT queries and prove their equivalence. We first define a *high-level calculus* as a mere application of the POIM transformation. Then we propose a *low-level calculus* which is defined by means of several applications of the POIM transformation followed by a "merging" process. Both calculi implement faithfully the SPARQL semantics for CONSTRUCT queries (Theorem 19). In Section 5, we show how the POIM transformation can be used to define a novel operational semantics of the SELECT queries. This semantics, which is faithful to SPARQL definitions (Theorem 37), is obtained by an original translation of each SELECT query into a CONSTRUCT query. Concluding remarks and related work are discussed in Section 6. The missing proofs are in the Appendix.

2 Graphs of triples

The set of IRIs, denoted Iri, and the set of literals, denoted Lit, with its usual operations, are defined in [17]. The sets Iri and Lit are disjoint. In addition, let B be a countably infinite set, disjoint from Iri and Lit. The elements of B are called the *blanks*. According to [17, Section 3.1], *an RDF graph is a set of RDF triples and an RDF triple consists of three components: the subject, which is an IRI or a blank node; the predicate, which is an IRI; and the object, which is an IRI, a literal or a blank node. The set of nodes of an RDF graph is the set of subjects and objects of triples in the graph.* Using set-theoretic notations,

this can be expressed as follows: let $Tr = (Iri \cup B) \times Iri \times (Iri \cup Lit \cup B)$, then an RDF triple is an element of Tr and an RDF graph is a subset of Tr. Let us also consider the following extension of RDF [17, Section 7]: *A generalized RDF triple is a triple having a subject, a predicate, and object, where each can be an IRI, a blank node or a literal. A generalized RDF graph is a set of generalized RDF triples.* Let $I = Iri \cup Lit$, so that a generalized RDF triple is an element of $(I \cup B)^3$ and a generalized RDF graph is a subset of $(I \cup B)^3$.

Let V be a countably infinite set disjoint from Iri, Lit and B. The elements of V are called the *variables*. According to [16, Section 2] *a set of triple patterns is called a basic graph pattern. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable.* Let $Tr_V = (Iri \cup B \cup V) \times (Iri \cup V) \times (Iri \cup Lit \cup B \cup V)$, then a triple pattern is an element of Tr_V and a basic graph pattern is a subset of Tr_V. Since Tr_V is a subset of $(I \cup B \cup V)^3$, each basic graph pattern is a subset of $(I \cup B \cup V)^3$.

RDF graphs and basic graph patterns are generalized in Definition 2 as *data graphs* and *query graphs* respectively, both relying on Definition 1.

Definition 1. *For each set A, the* triples on A *are the elements of A^3. For each triple $t = (s, p, o)$ on A the elements s, p and o of A are called respectively the* subject*, the* predicate *and the* object *of t. A* graph on A *is a set of triples on A, i.e. a subset of A^3. For each graph T on A, the subset of A made of the subjects, predicates and objects of T is called the set of* attributes *of T and is denoted $|T|$; it follows that T is a subset of $|T|^3$. Let T and T' be two graphs on A. A* morphism $a : T \to T'$ *is a map such that there is a map $M : |T| \to |T'|$ such that a is the restriction of M^3 to T. Then M is uniquely determined by a, it is denoted $|a|$. This yields the* category of graphs on A*, denoted $\mathcal{G}(A)$. We say that a morphism $a : T \to T'$ of graphs on A* fixes *a subset C of A if $|a|(x) = x$ for each x in $|T| \cap C$. For each subset C of A, the subcategory of $\mathcal{G}(A)$ made of the graphs on A with the morphisms fixing C is denoted $\mathcal{G}_C(A)$.*

Thus, by mapping a to $|a|$ we get a one-to-one correspondence between the morphisms $a : T \to T'$ of graphs on A and the maps $M : |T| \to |T'|$ such that $M^3(T) \subseteq T'$. An isomorphism (i.e., an invertible morphism) in $\mathcal{G}(A)$ is a morphism $a : T \to T'$ of graphs on A such that $|a| : |T| \to |T'|$ is a bijection and $a(T) = T'$. A morphism a fixing C is determined by the restriction of the map $|a|$ to $|T| \cap \overline{C}$, where $\overline{C} = A \setminus C$. An isomorphism a in $\mathcal{G}_C(A)$ is a morphism $a : T \to T'$ of graphs on A such that $|a|$ is the identity on $|T| \cap C$ and a bijection between $|T| \cap \overline{C}$ and $|T'| \cap \overline{C}$ and $a(T) = T'$. The notions of *inclusion*, *subgraph*, *image* and *union* for graphs on A are defined as inclusion, subset, image and union for subsets of A^3.

Definition 2. *Let I, B and V be three pairwise distinct countably infinite sets, called respectively the sets of* resource identifiers*,* blanks *and* variables*. Let $IB = I \cup B$, $IV = I \cup V$ and $IBV = I \cup B \cup V$. The* category of data graphs *is $\mathcal{D} = \mathcal{G}(IB)$ and for each subset C of IB the category of data graphs* fixing *C is the subcategory $\mathcal{D}_C = \mathcal{G}_C(IB)$ of $\mathcal{D}$. The* category of query graphs *is*

$\mathcal{Q} = \mathcal{G}(IBV)$ *and for each subset* C *of* IBV *the category of query graphs* fixing C *is the subcategory* $\mathcal{Q}_C = \mathcal{G}_C(IBV)$ *of* $\mathcal{Q}$.

Thus, when $I = Iri \cup Lit$, the RDF graphs are the data graphs where only nodes can be blanks and only nodes that are not subjects can be literals, and the RDF terms of an RDF graph are its attributes when it is seen as a data graph. Then the isomorphisms of RDF graphs, as defined in [17, Section 3.6.], are the isomorphisms in the category $\mathcal{D}_I$ of data graphs fixing I: indeed, two data graphs G_1 and G_2 are isomorphic in $\mathcal{D}_I$ if and only if they differ only by the names of their blanks. For each data graph T, let $|T|_I = |T| \cap I$ and $|T|_B = |T| \cap B$, so that $|T|$ is the disjoint union of $|T|_I$ and $|T|_B$. Similarly, the basic graph patterns of SPARQL are the query graphs where only nodes can be blanks and only nodes that are not subjects can be literals. For each query graph T, let $|T|_I = |T| \cap I$, $|T|_B = |T| \cap B$ and $|T|_V = |T| \cap V$, so that $|T|$ is the disjoint union of $|T|_I$, $|T|_B$ and $|T|_V$.

Morphisms of graphs can be used, for instance, for substituting the variables of a query graph (Definition 3) or for interpreting a data graph in a universe of discourse (Definition 4).

Definition 3. *A* match *from a query graph* L *to a data graph* G *is a morphism of query graphs from* L *to* G *which fixes* I*. The set of matches from* L *to* G *is denoted* $\mathcal{M}atch(L, G)$ *and the set of all matches from* L *to any data graph is denoted* $\mathcal{M}atch(L)$.

Thus, a match fixes each resource identifier and it maps a variable or a blank to a resource identifier or a blank.

The interpretations of an RDF graph are also kinds of morphisms, see Definition 4. Note that this will not be used later in this paper. We define an interpretation of a data graph G in a universe of discourse U by generalizing the definition of a morphism, according to [17, Section 1.2.]: *Any IRI or literal denotes something in the world (the "universe of discourse"). These things are called resources. The predicate itself is an IRI and denotes a property, that is, a resource that can be thought of as a binary relation.* Recall that the binary relations on a set R are the subsets of R^2. It can happen that a binary relation on R is itself an element of R.

Definition 4. *Given a set* R *and a subset* P *of* R^2 *made of binary relations on* R*, let* U *be the set of triples* (s, p, o) *in* R^3 *such that* $p \in P$ *and* $(s, o) \in p$*. The* universe of discourse *with* R *as set of* resources *and* P *as set of* properties *is the graph* U *on* R*. Given a universe of discourse* U *on a set* R *and a map* $M_I : I \to R$*, an* interpretation *of a data graph* G *is a map* $i : G \to U$ *such that* $i = M^3$ *for a map* $M : |G| \to |U|$ *which extends* M_I.

In this paper, we consider categories $\mathcal{D}_C$ and $\mathcal{Q}_C$ for various subsets C of IB and IBV respectively. It will always be the case that C contains I, so that we can say that resource identifiers have a "global scope". In contrast, blanks have a "local scope": in the basic part of RDF and SPARQL considered in this

paper, the scope of a blank node is restricted to one data graph or one query graph. The note about *blank node identifiers* in [17, Section 3.4] distinguishes two kinds of syntaxes for RDF: an abstract syntax where blank nodes do not have identifiers and concrete syntaxes where blank nodes have identifiers. In our approach a blank *is* an attribute, which corresponds to a concrete syntax, and the abstract syntax is obtained by considering data graphs as objects of the category $\mathcal{D}_I$ up to isomorphism, so that any blank node can be changed for a new blank node if needed.

Example 5. In all examples we use the following prefixes (`@prefix` for data and `PREFIX` for queries):

Prefixes

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/>.
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
```

Consider two RDF graphs G_1, G_2 as follows. They are isomorphic in $\mathcal{D}_I$ but not in $\mathcal{D}_{IB}$ because blanks are swapped.

G_1

```
<http://example.org/Al> foaf:knows _:b.
_:c foaf:knows <http://example.org/Bob>.
```

G_2

```
<http://example.org/Al> foaf:knows _:c.
_:b foaf:knows <http://example.org/Bob>.
```

Now consider basic graph patterns G_3 to G_8. They are pairwise non-isomorphic in $\mathcal{Q}_{IBV}$ because they are pairwise distinct. In $\mathcal{Q}_{IV}$ only G_7 and G_8 are isomorphic. In $\mathcal{Q}_I$ these query graphs belong to two different isomorphism classes: on one side G_3 and G_4 are isomorphic and on the other side G_5, G_6, G_7 and G_8 are isomorphic.

G_3

```
<http://example.org/Al> foaf:knows _:b.
_:b foaf:knows <http://example.org/Bob>.
```

G_4

```
<http://example.org/Al> foaf:knows ?x.
?x foaf:knows <http://example.org/Bob>.
```

G_5

```
<http://example.org/Al> foaf:knows _:b.
_:c foaf:knows <http://example.org/Bob>.
```

G_6

```
<http://example.org/Al> foaf:knows ?x.
?y foaf:knows <http://example.org/Bob>.
```

G_7

```
<http://example.org/Al> foaf:knows ?x.
_:b foaf:knows <http://example.org/Bob>.
```

G_8

```
<http://example.org/Al> foaf:knows ?x.
_:c foaf:knows <http://example.org/Bob>.
```

Assumption 6 *From now on A is a set, C is a subset of A, $\overline{C} = A \setminus C$ is the complement of C in A, and it is assumed that both C and $\overline{C}$ are countably infinite.*

Remark 7. Since $\overline{C}$ is countably infinite, when dealing with a finite number of finite graphs on A it is always possible to find a *new attribute outside* C, i.e., an element of $\overline{C}$ that is not an attribute of any of the given graphs. We will use repeatedly the following consequence of this fact:

Given a graph T on A, if any attribute of T in $\overline{C}$ is replaced by any new element of $\overline{C}$ the result is a graph T' on A that is isomorphic to T in $\mathcal{G}_C(A)$. Such a T' exists when T is finite.

Now let us focus on some kinds of colimits of graphs on A: coproducts in Proposition 8 and pushouts in Proposition 9. Recall that colimits in any category are defined up to isomorphism in this category.

Proposition 8. *Given graphs $T_1, ..., T_k$ on A such that $|T_i| \cap |T_j| \subseteq C$ for each $i \neq j$, the union $T_1 \cup ... \cup T_k$ is a coproduct of $T_1, ..., T_k$ in $\mathcal{G}_C(A)$.*

By Remark 7 it follows that if $T_1, ..., T_k$ are any finite graphs on A there are graphs $T'_1, ..., T'_k$ on A such that T'_i is isomorphic to T_i in $\mathcal{G}_C(A)$ for each i and $|T'_i| \cap |T'_j| \subseteq C$ for each $i \neq j$, so that the union $T'_1 \cup ... \cup T'_k$ is a coproduct of $T_1, ..., T_k$ in $\mathcal{G}_C(A)$.

Proposition 9. *Let $l : L \to K$ and $m : L \to G$ be morphisms of graphs on A such that K is finite, l is an inclusion and m fixes C. Let us assume that $|K| \cap |G| \subseteq C$ (this is always possible up to isomorphism in $\mathcal{G}_C(A)$, by Remark 7). Let $N : |K| \to |G| \cup |K \setminus L|$ be such that $N(x) = |m|(x)$ for $x \in |L|$ and $N(x) = x$ otherwise. Let $D = G \cup N^3(K)$, let $n : K \to D$ be the restriction of N^3 and $g : G \to D$ the inclusion. Then $|D| = |G| \cup |K \setminus L|$ and the square (l, m, n, g) is a pushout square in $\mathcal{G}_C(A)$.*

Thus, D is a kind of "union of G and K over L", however in general it is *not* the case that D is the union of G and $K \setminus L$. It is the case that $D = G \cup D_2$ where $D_2 = N^3(K \setminus L)$ but N^3 is not the identity on $K \setminus L$, and moreover G and D_2 are not disjoint in general.

3 The POIM transformation

A SPARQL query like "CONSTRUCT $\{R\}$ WHERE $\{L\}$" is called *basic* when both R and L are basic graph patterns. In such a query, variables with the same name in L and R denote the same RDF term, whereas it is not the case for blank nodes. The statement "*blank nodes in graph patterns act as variables*" in [16, Section 4.1.4] holds for L, whereas blank nodes in R give rise to new blank nodes in the result of the query as in Examples 17 and 21. Thus, the meaning of blank nodes in L is unrelated to the meaning of blank nodes in R, and in both L and R each blank can be replaced by a new blank.

We generalize this situation in Definition 10 by allowing any data graphs for L and R up to isomorphism in $\mathcal{Q}_{IV}$: the resource identifiers and the variables in L and R are fixed but each blank can be replaced by a new blank. Thus, without loss of generality, we can assume that $|L|_B \cap |R|_B = \emptyset$. Under this assumption, the set of triples $K = L \cup R$ with the inclusions of L and R in K is a coproduct of L and R in the category $\mathcal{Q}_{IV}$. We also assume that each variable in R occurs in L, so that every substitution for the variables in L provides a substitution for the variables in R. This assumption $|R|_V \subseteq |L|_V$ is equivalent to $|K|_V = |L|_V$.

Definition 10. *A* basic construct query *is a pair of finite query graphs (L, R) such that $|L|_B \cap |R|_B = \emptyset$ and $|R|_V \subseteq |L|_V$, up to isomorphism in the category $\mathcal{Q}_{IV}$. The* transformation rule *of a basic construct query (L, R) is the cospan*

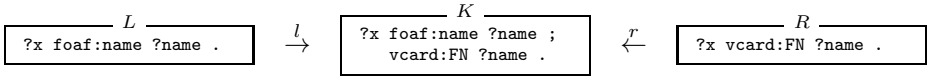
$P_{L,R} = (L \xrightarrow{l} K \xleftarrow{r} R)$ *where* $K = L \cup R$ *and* l *and* r *are the inclusions. Its* left-hand side *is* L *and its* right-hand side *is* R.

$$P_{L,R} \;=\; L \xrightarrow[\subseteq]{l} K = L \cup R \xleftarrow[\supseteq]{r} R$$

Example 11. Consider the following SPARQL CONSTRUCT query:

Query
```
CONSTRUCT { ?x vcard:FN ?name } WHERE { ?x foaf:name ?name }
```

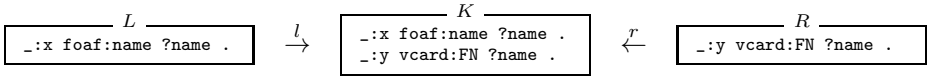
In the corresponding transformation rule $L \xrightarrow{l} K \xleftarrow{r} R$ there are no blanks in L nor in R, thus the transformation rule is $L \xrightarrow{l} K \xleftarrow{r} R$ where l and r are the inclusions of L and R in $K = L \cup R$.



Example 12. Now the SPARQL CONSTRUCT query from Example 11 is modified by replacing both occurrences of the variable `?x` by the blank node `_:x`:

Query
```
CONSTRUCT { _:x vcard:FN ?name } WHERE { _:x foaf:name ?name }
```

In the corresponding transformation rule one blank has been modified so as to ensure that $|L|_B \cap |R|_B$ is empty:



When a basic SPARQL query "CONSTRUCT $\{R\}$ WHERE $\{L\}$" is run against an RDF graph G, and when there is precisely one match of L into G, the result of the query is an RDF graph H obtained by substituting the variables in R. This substitution can be seen as a match of R into H. We claim that the process of building H with this match of R into H from the match of L into G can be seen as a two-step process involving an intermediate match of K in an RDF graph D. The definition of this process relies on an algebraic construction that we call the *POIM transformation*: PO for *pushout* and IM for *image* (Definition 13). The POIM transformation is related to a large family of algebraic graph transformations based on pushouts, like the SPO (Simple Pushout) [9], DPO (Double Pushout) [8] or SqPO (Sesqui-Pushout) [7].

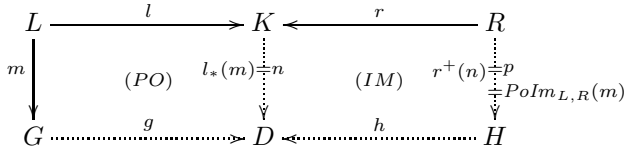
Given a basic construct query (L, R) and its transformation rule $L \xrightarrow{l} K \xleftarrow{r} R$, the POIM transformation is defined as a map from the matches of L to the matches of R, in two steps: first from the matches of L to the matches of K, then from the matches of K to the matches of R. Given an inclusion $l : L \to K$ in $\mathcal{Q}_I$, the *cobase change along* l is the map $l_* : \mathcal{M}atch(L) \to \mathcal{M}atch(K)$ that maps each $m : L \to G$ to $l_*(m) : K \to D$ defined from the pushout of l and m in $\mathcal{Q}_I$, as described in Proposition 9. Note that D is a data graph because of the assumption $|K|_V = |L|_V$. Given an inclusion $r : R \to K$ in $\mathcal{Q}_I$, the *image factorization along* r is the map $r^+ : \mathcal{M}atch(K) \to \mathcal{M}atch(R)$ that maps each

$n : K \to D$ to $r^+(n) : R \to H$ where H is the image of R in D and $r^+(n)$ is the restriction of n and $h : H \to D$ is the inclusion. This leads to Definition 13 and Proposition 14.

Definition 13. *Let (L, R) be a basic construct query and $L \xrightarrow{l} K \xleftarrow{r} R$ its transformation rule. The* POIM transformation map *of (L, R) is the map*

$$PoIm_{L,R} = r^+ \circ l_* : \mathcal{M}atch(L) \to \mathcal{M}atch(R)$$

composed of the cobase change map l_* and the image factorization map r^+. The result *of applying $PoIm_{L,R}$ to a match $m : L \to G$ is the match $PoIm_{L,R}(m) : R \to H$ or simply the query graph H.*



(1)

Note that the result H is defined only up to isomorphism in $\mathcal{Q}_I$, which means that the blanks in H can be modified (as long as this modification does not identify any of them).

Proposition 14. *Let (L, R) be a basic construct query and $m : L \to G$ a match. Let $P : |R| \to A$ be defined by $P(x) = |m|(x)$ for $x \in |R|_V$ and $P(x) = x$ otherwise. Then, up to isomorphism in $\mathcal{Q}_I$, the result of applying $PoIm_{L,R}$ to m is $p : R \to H$ where $H = P^3(R)$ and p is the restriction of P^3.*

Remark 15. Each set $\mathcal{M}atch(X)$ can be seen as a coslice category, then the maps r^+ and l_* can be seen as functors: this could be useful when extending this paper to additional features of SPARQL.

Example 16. Consider the SPARQL CONSTRUCT query from Example 11:

Query
```
CONSTRUCT { ?x vcard:FN ?name } WHERE { ?x foaf:name ?name }
```

and let us run this query against the RDF graph G:

G
```
ex:a foaf:name "Alice" ; foaf:nick "Lissie" .
```

There is a single match m, it is such that $m(?\texttt{x}) = \texttt{ex:a}$ and $m(\texttt{?name}) = \texttt{"Alice"}$. The POIM transformation produces successively the following data graphs D and H, where H is the query result:

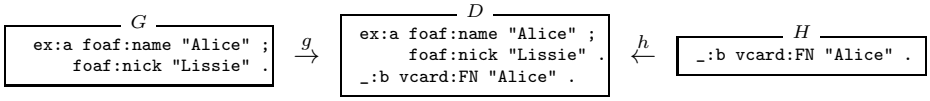
G
```
ex:a foaf:name "Alice" ;
     foaf:nick "Lissie" .
```
$\xrightarrow{g}$
D
```
ex:a foaf:name "Alice" ;
     foaf:nick "Lissie" ;
     vcard:FN "Alice" .
```
$\xleftarrow{h}$
H
```
ex:a vcard:FN "Alice" .
```

Example 17. Now consider the SPARQL CONSTRUCT query from Example 12:

Query

```
CONSTRUCT { _:x vcard:FN ?name } WHERE { _:x foaf:name ?name }
```

Let us run this query against the RDF graph G from Example 16. There is a single match m, it is such that $m(\texttt{\_:x}) = \texttt{ex:a}$ and $m(\texttt{?name}) = \texttt{"Alice"}$. The POIM transformation produces successively the following data graphs D and H, where H is the query result:



4 Running basic construct queries

In Section 3, we defined the POIM transformation and we applied it to run a basic construct query (L, R) against a data graph G, under the assumption that there is exactly one match from L to G. Now we define two different calculi for running a basic construct query against a data graph G without any assumption on the number of matches. The *high-level calculus* (Definition 22) is one "large" application of the POIM transformation. The *low-level calculus* (Definition 24) consists of several "small" applications of the POIM transformation followed by a "merging" process. In Propositions 23 and 27 we prove that both calculi return the same result. This result coincides (up to the renaming of the blanks) with the result returned by SPARQL when L and R are basic graph patterns and G is an RDF graph (Theorem 19).

Definition 18. *Let (L, R) be a basic construct query and G a data graph. Assume (without loss of generality) that $|G|_B \cap |L|_B = \emptyset$ and $|G|_B \cap |R|_B = \emptyset$. Let $m_1, ..., m_k$ be the matches from L to G. For each $i = 1, ..., k$ let H_i be the data graph obtained from R by replacing each variable x in R by $m_i(x)$ and each blank in R by a new blank (which means: a new blank for each blank in R and each i in $\{1, ..., k\}$). The* query result *of applying the basic construct query (L, R) to the data graph G is the data graph $H = H_1 \cup ... \cup H_k$.*

A triple (s, p, o) in $(I \cup B)^3$, where $I = Iri \cup Lit$, is *well-formed* if it is an RDF triple, in the sense that $s \in Iri \cup B$ and $p \in Iri$. Thus, a data graph is an RDF graph if and only if all its triples are well-formed. The *answer* of a SPARQL CONSTRUCT query over an RDF graph is defined in [13].

Theorem 19. *Let L and R be basic graph patterns with $|L|_B = \emptyset$ and $|R|_V \subseteq |L|_V$. Then (L, R) is a basic construct query and the set of well-formed triples in the query result of applying (L, R) to an RDF graph G is isomorphic in $\mathcal{D}_I$ to the answer of the SPARQL query "CONSTRUCT $\{R\}$ WHERE $\{L\}$" over G.*

Example 20. Consider the SPARQL query from Examples 11 and 16:

Query

```
CONSTRUCT { ?x vcard:FN ?name } WHERE { ?x foaf:name ?name }
```

and let us run this query against the RDF graph G:

G
```
ex:a foaf:name "Alice" ; foaf:nick "Lissie" .
ex:b foaf:name "Bob" ; foaf:nick "Bobby" .
```

There are two matches and we get the RDF graphs H_1, H_2 and the result H:

H_1
```
ex:a vcard:FN "Alice" .
```

H_2
```
ex:b vcard:FN "Bob" .
```

H
```
ex:a vcard:FN "Alice" .
ex:b vcard:FN "Bob" .
```

Example 21. Consider the SPARQL CONSTRUCT query:

Query
```
CONSTRUCT { _:c vcard:FN ?name } WHERE { ?x foaf:name ?name }
```

Note that this query always returns the same result as the query from Examples 12 and 17. Let us run this query against the RDF graph G from Example 20. There are two matches and we get the RDF graphs H_1, H_2 and the result H:

H_1
```
_:c vcard:FN "Alice" .
```

H_2
```
_:c vcard:FN "Bob" .
```

H
```
_:c1 vcard:FN "Alice" .
_:c2 vcard:FN "Bob" .
```

Let k be a natural number. According to Proposition 8, for each query graph T the query graph $k\,T$, coproduct of k copies of T in $\mathcal{Q}_I$, can be built (up to isomorphism) as follows: for each $i \in \{1, ..., k\}$ let T_i be a copy of T where each blank and variable has been renamed in such a way that there is no blank or variable common to two of the T_i's, then the query graph $k\,T$ is the union $T_1 \cup ... \cup T_k$. Now let (L, R) be a basic construct query. The transformation rule $P_{L,R} = (L \stackrel{l}{\rightarrow} K \stackrel{r}{\leftarrow} R)$ is a cospan in $\mathcal{Q}_I$, that gives rise to the cospan $k\,P_{L,R} = (k\,L \stackrel{k\,l}{\rightarrow} k\,K \stackrel{k\,r}{\leftarrow} k\,R)$. Since l and r are inclusions, this renaming can be done simultaneously in the copies of L, K and R, so that $k\,K = k\,L \cup k\,R$ and $k\,l$ and $k\,r$ are the inclusions. Thus, $(k\,L, k\,R)$ is a basic construct query and $P_{k\,L,k\,R} = k\,P_{L,R}$ is its corresponding transformation rule.

Definition 22. *Let (L, R) be a basic construct query and G a data graph. Let $m_1, ..., m_k$ be the matches from L to G. Consider the basic construct query $(k\,L, k\,R)$. Let m be the match from $k\,L$ to G that coincides with m_i on the i-th component of $k\,L$. The* high-level query result of (L, R) against G *is the result H_{high} of applying the POIM transformation map $PoIm_{k\,L,k\,R}$ to the match $m : k\,L \to G$, as in Diagram (2).*

$$\begin{array}{ccccc} k\,L & \xrightarrow{k\,l} & k\,K & \xleftarrow{k\,r} & k\,R \\ {\scriptstyle m}\downarrow & (PO) & \downarrow{\scriptstyle n} & (IM) & \downarrow{\scriptstyle p} \\ G & \xrightarrow[g]{} & D & \xleftarrow[h]{} & H_{high} \end{array} \qquad (2)$$

Proposition 23. *Let (L, R) be a basic construct query and G a data graph. The high-level query result of (L, R) against G is isomorphic, in the category $\mathcal{D}_I$, to the query result of (L, R) against G.*

The low-level calculus is a two-step process: first one *local result* is obtained for each match, using a POIM transformation, then the local results are glued together in order to form the *low-level query result*.

Definition 24. *Let (L, R) be a basic construct query and G a data graph. Let $m_1, ..., m_k$ be the matches from L to G. For each $i = 1, ..., k$ let G_i be the image of m_i and let us still denote m_i the restriction $m_i : L \to G_i$. The* local result H_i *of (L, R) against G along m_i is the result of applying the POIM transformation map $PoIm_{L,R}$ to the match $m_i : L \to G_i$. Let $IB(G) = I \cup |G|_B$. The* low-level query result H_{low} *of (L, R) against G is the coproduct of the H_i's in the category $\mathcal{D}_{IB(G)}$ of data graphs with morphisms fixing all resource identifiers and the blanks that are in G.*

$$\begin{array}{ccccc} L & \xrightarrow{\quad l \quad} & K & \xleftarrow{\quad r \quad} & R \\ {\scriptstyle m_i}\downarrow & (PO) & {\scriptstyle n_i}\downarrow & (IM) & \downarrow{\scriptstyle p_i} \\ G_i & \xrightarrow[g_i]{} & D_i & \xleftarrow[h_i]{} & H_i \end{array} \tag{3}$$

Example 25. Let us apply the low-level calculus to Example 21. The match m_1 produces $G_1 \to D_1 \leftarrow H_1$:

G_1
```
ex:a foaf:name "Alice" .
```
$\xrightarrow{g_1}$ D_1
```
ex:a foaf:name "Alice" .
_:c vcard:FN "Alice" .
```
$\xleftarrow{h_1}$ H_1
```
_:c vcard:FN "Alice" .
```

and similarly the match m_2 produces $G_2 \to D_2 \leftarrow H_2$:

G_2
```
ex:b foaf:name "Bob" .
```
$\xrightarrow{g_2}$ D_2
```
ex:b foaf:name "Bob" .
_:c vcard:FN "Bob" .
```
$\xleftarrow{h_2}$ H_2
```
_:c vcard:FN "Bob" .
```

Finally the query result H_{low}, which is the coproduct of H_1 and H_2 in category $\mathcal{D}_{IB(G)}$, is isomorphic to H from Example 21.

H_{low}
```
_:c1 vcard:FN "Alice" . _:c2 vcard:FN "Bob" .
```

Example 26. This example illustrates how local results are "merged" to compute the result in the low-level calculus. The SPARQL query is the following:

Query
```
CONSTRUCT { ?x rel:acquaintanceof ?z . } WHERE { ?x foaf:knows ?y . ?y foaf:knows ?z . }
```

Its corresponding transformation rule is:

L
```
?x foaf:knows ?y .
?y foaf:knows ?z .
```
$\xrightarrow{l}$ K
```
?x foaf:knows ?y ;
   rel:acquaintanceOf ?z .
?y foaf:knows ?z .
```
$\xleftarrow{r}$ R
```
?x rel:acquaintanceOf ?z .
```

This query is applied to the following graph G:

G
```
<http://example.org/Alice> foaf:knows <http://example.org/Bob> .
<http://example.org/Bob> foaf:knows _:c .
_:c foaf:knows <http://example.org/Alice> .
```

There are three matches m_1, m_2, m_3, thus three local results H_1, H_2, H_3:

H_1

```
<http://example.org/Alice>  rel:acquaintanceOf _:c .
```

H_2

```
_:c rel:acquaintanceOf <http://example.org/Bob> .
```

H_3

```
<http://example.org/Bob>  rel:acquaintanceOf <http://example.org/Alice> .
```

The blank `_:c` in H_1 and H_2 is not duplicated in the coproduct H_{low} because it comes from G. Thus the result is:

H_{low}

```
<http://example.org/Alice>  rel:acquaintanceOf _:c .
_:c rel:acquaintanceOf <http://example.org/Bob> .
<http://example.org/Bob>  rel:acquaintanceOf <http://example.org/Alice> .
```

Proposition 27. *Let (L, R) be a basic construct query and G a data graph. The low-level query result of (L, R) against G is isomorphic, in the category $\mathcal{D}_I$, to the query result of (L, R) against G.*

5 Running basic select queries

The CONSTRUCT query form of SPARQL returns a data graph whereas the SELECT query form returns a table, like the SELECT query form of SQL. Both in SQL and in SPARQL, it is well-known that such tables are not exactly relations in the mathematical sense: in mathematics a relation on $X_1, ..., X_n$ is a subset of the cartesian product $X_1 \times ... \times X_n$, while the result of a SELECT query in SQL or SPARQL is a multiset of elements of $X_1 \times ... \times X_n$. In order to avoid ambiguities, such a multiset is called a *multirelation* on $X_1, ..., X_n$. When all X_i's are the same set X it is called a multirelation of arity n on X.

A SPARQL query such as "SELECT $?s_1, ..., ?s_n$ WHERE $\{L\}$" is called *basic* when L is a basic graph pattern and $?s_1, ..., ?s_n$ are distinct variables. We generalize this situation by defining a *basic select query* as a pair (L, S) where L is a finite query graph and S is a finite set of variables. Then we associate to each basic select query (L, S) a basic construct query $(L, Gr(S))$. Finally we define the result of running the basic select query (L, S) against a data graph G from the data graph H result of running the basic construct query $(L, Gr(S))$ against G. This process is first described on an example.

Example 28. Consider the following SPARQL SELECT query:

SELECT Query

```
SELECT ?nameX ?nameY
WHERE{ ?x foaf:knows ?y ; foaf:name ?nameX . ?y foaf:name ?nameY . }
```

We associate to this SELECT query the following CONSTRUCT query:

CONSTRUCT Query

```
CONSTRUCT { _:r <http://example.org/nameX> ?nameX ; <http://example.org/nameY> ?nameY . }
WHERE { ?x foaf:knows ?y ; foaf:name ?nameX . ?y foaf:name ?nameY . }
```

Let us run this CONSTRUCT query against the RDF graph G:

G

```
_:a foaf:name "Alice" ; foaf:knows _:b ; foaf:knows _:c .
_:b foaf:name "Bob" .
_:c foaf:name "Cathy" .
```

The result is the RDF graph H:

H

```
_:l1 <http://example.org/nameX> "Alice" ; <http://example.org/nameY> "Bob" .
_:l2 <http://example.org/nameX> "Alice" ; <http://example.org/nameY> "Bob" .
_:l3 <http://example.org/nameX> "Alice" ; <http://example.org/nameY> "Cathy" .
```

From the RDF graph H we get the following table, by considering each blank $\_:l_i$ in H as the identifier of a line in the table. Note that the set of triples in H becomes a multiset of lines in the table. This table is indeed the answer of the SPARQL SELECT query over G.

nameX	nameY
"Alice"	"Bob"
"Alice"	"Bob"
"Alice"	"Cathy"

In order to generalize Example 28 we have to define a transformation from each SELECT query to a CONSTRUCT query and a transformation from the result of this CONSTRUCT query to the result of the given SELECT query. For this purpose, we first define *relational* data graphs (Definition 29) and *relational* query graphs (Definition 32).

Definition 29. *A* relational data graph *on a finite set* $\{s_1, ..., s_n\}$ *of resource identifiers is a data graph made of triples* $(\_ : l_i, s_j, y_{i,j})$ *where the* $\_ : l_i$*'s are pairwise distinct blanks and the* $y_{i,j}$*'s are in* IB*, for* $j \in \{1, ..., n\}$ *and* i *in some finite set* $\{1, ..., k\}$*.*

Proposition 30.
Each relational data graph $S = \{(\_ : l_i, s_j, y_{i,j})\}_{i \in \{1,...,k\}, j \in \{1,...,n\}}$ *determines a multirelation* $Rel(S) = \{(y_{i,1}, ..., y_{i,n})\}_{i \in \{1,...,k\}}$ *of arity* n *on* IB*.*

Example 31. Here is a relational data graph on $\{\texttt{nameX}, \texttt{nameY}\}$ with its corresponding multirelation:

```
_:l1 nameX "Alice" ; nameY "Bob" .
_:l2 nameX "Alice" ; nameY "Cathy" .
_:l3 nameX "Alice" ; nameY "Cathy" .
```

nameX	nameY
"Alice"	"Bob"
"Alice"	"Cathy"
"Alice"	"Cathy"

Assume that each variable in SPARQL is written as "$?s$" for some string s.

Definition 32. *The* relational query graph *on a finite set of variables* $S = \{?s_1, ..., ?s_n\}$ *is the query graph* $Gr(S)$ *made of the triples* $(\_ : r, s_j, ?s_j)$ *where* $j \in \{1, ..., n\}$ *and* $\_ : r$ *is a blank. Note that* $Gr(S)$ *is uniquely determined by* S *up to isomorphism in* $\mathcal{Q}_{IV}$*.*

Example 33. Here is the relational query graph on $\{\texttt{?nameX}, \texttt{?nameY}\}$:

```
_:r nameX ?nameX ; nameY ?nameY .
```

In the following we show how a basic select query can be encoded as a basic construct query (Definition 34) and we prove that the result of the given select query is easily recovered from the result of its associated construct query (Theorem 37).

Definition 34. *A* basic select query *is a pair (L, S) where L is a finite query graph and S is a finite set of variables such that each variable in S occurs in L. The* basic construct query associated to *a basic select query (L, S) is $(L, Gr(S))$ where $Gr(S)$ is the relational query graph on S.*

Proposition 35. *Let (L, S) be a basic select query and G a data graph. The query result of $(L, Gr(S))$ against G is a relational data graph H. More precisely, let $S = \{?s_1, ..., ?s_n\}$ and let $m_1, ..., m_k$ be the matches from L to G, then H is the set of triples $(\_ : l_i, s_j, m_i(?s_j))$ where $i \in \{1, ..., k\}$, $j \in \{1, ..., n\}$, and the blanks $\_ : l_1, ..., \_ : l_k$ are pairwise distinct.*

Because of Proposition 35 we can state the following definition.

Definition 36. *Let (L, S) be a basic select query and G a data graph. Let H be the query result of $(L, Gr(S))$ against G. The* query result *of (L, S) against G is the multirelation $Rel(H)$ on IB.*

Theorem 37. *Let L be a basic graph pattern of SPARQL and $S = \{?s_1, ..., ?s_n\}$ a finite set of variables included in $|L|_V$ and let G be an RDF graph. Then the query result of (L, R) against G is the answer of the SPARQL query "SELECT $?s_1, ..., ?s_n$ WHERE $\{L\}$" over G.*

Remark 38. This Section can be generalized to multirelations with "null" values (as in SQL) by allowing some missing triples in the definition of relational data graphs.

6 Conclusion

Relational algebra [6] is the main mathematical foundation underlying SQL-like formalism for databases. However new frameworks such as RDF and SPARQL, where data structures are represented as graphs, are better adapted to the needs of big data and web applications. So, new mathematical foundations are needed to cope with this change in data encodings, see e.g., [4, 14, 11].

In this paper, we make the bet to base our work entirely on algebraic theories behind graphs and their transformations. Suitable categories of data graphs and query graphs are defined and the definition of morphisms of query graphs clarifies the difference between blank nodes and variables. Besides, we propose to encode CONSTRUCT and SELECT queries as graph rewrite rules, of the form $L \to L \cup R \leftarrow R$, and define their operational semantics following a novel algebraic approach called POIM. From the proposed semantics, blanks in L play the

same role as variables and thus can be replaced by variables, whereas blanks in R are used for creating new blanks in the result of a CONSTRUCT query. As in [13], we focus on the CONSTRUCT query form as the fundamental query form. In addition we propose a translation of the SELECT queries as CONSTRUCT queries compatible with their operational semantics. One of the benefits of using category theory is that coding of data graphs as sets of triples is not that important. The results we propose hold for all data models which define a category with enough colimits. For intance, one may expect to define data graph categories for the well-known Edge-labelled graphs or Property graphs [15]. The proposed operational semantics can clearly benefit from all results regarding efficient graph matching implementation, see e.g. [10].

Among related works, a category of RDF-graphs as well as their transformations have been proposed in [5]. The authors defined objects of RDF-Graph categories of the form $(G_{\text{Blank}}, G_{\text{Triple}})$ where G_{Blank} and G_{Triple} denote respectively the set of blank nodes and the set of triples of graph G. This definition is clearly different from ours (Definition 2). In addition, the morphisms of such RDF-graphs associate blank nodes to blank nodes which is not always the case in our approach. Associating a blank node to any element of a triple is called instantiation in [5]. The authors did not tackle the problem of answering SPARQL queries but rather proposed an algebraic approach to transform RDF-graphs. Their approach, called MPOC-PO, is inspired from DPO where the first square is replaced by a "minimal" pushout complement (MPOC). MPOC-PO drastically departs from the POIM transformations we propose. This difference is quite natural since the two approaches have different objectives : the POIM approach is dedicated to implement SPARQL queries while the MPOC-PO is intended to transform RDF-graphs in general. However, MPOC-PO and DPO approaches are clearly not tailored to implement CONSTRUCT or SELECT queries since the (minimal) pushout complements always include parts of large data graphs which are not matched by the queries while such parts are not involved in the query answers.

In [1], even if the authors use a categorical setting, their objectives and results depart from ours as they mainly encode every ontology as a category. However, Graph Transformations have already been used in modeling relational databases, see e.g. [3] where a visual and textual hybrid query language has been proposed. In [12], the main features of a data management system based on graphs have been proposed where the underlying typed attributed data graphs are different from those of RDF and SPARQL. In [2], triple graph grammars (TGG) have also been used for data modelling and model transformation rules to be compiled into Graph Data Bases code for execution.

In this paper we consider basic graphs and queries, which form a significant kernel of RDF and SPARQL. Future work includes the generalization of the present work to other features of RDF and SPARQL in order to encompass general SPARQL queries. We also consider studying RDF Schema [18] and ontologies from this point of view.

References

1. Aliyu, S., Junaidu, S., Kana, A.F.D.: A category theoretic model of RDF ontology. International Journal of Web & Semantic Technology (IJWesT) (2015)
2. Alqahtani, A., Heckel, R.: Model based development of data integration in graph databases using triple graph grammars. In Software Technologies: Applications and Foundations. Lecture Notes in Computer Science, vol. 11176, pp. 399–414. Springer (2018), https://doi.org/10.1007/978-3-030-04771-9_29
3. Andries, M., Engels, G.: A hybrid query language for an extended entity-relationship model. J. Vis. Lang. Comput. 7(3), 321–352 (1996), https://doi.org/10.1006/jvlc.1996.0017
4. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. 50(5), 68:1–68:40 (2017), https://doi.org/10.1145/3104031
5. Braatz, B., Brandt, C.: Graph transformations for the resource description framework. ECEASST 10 (2008), https://doi.org/10.14279/tuj.eceasst.10.158
6. Codd, E.F.: The relational Model for Database Management (Version 2 ed.). Addison-Wesley (1990)
7. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-pushout rewriting. In: ICGT 2006. LNCS, vol. 4178, pp. 30–45. Springer (2006)
8. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation - part I: basic concepts and double pushout approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars. pp. 163–246. World Scientific (1997)
9. Ehrig, H., Heckel, R., Korff, M., Löwe, M., Ribeiro, L., Wagner, A., Corradini, A.: Algebraic approaches to graph transformation - part II: single pushout approach and comparison with double pushout approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, pp. 247–312. World Scientific (1997)
10. Fan, W., Li, J., Ma, S., Wang, H., Wu, Y.: Graph homomorphism revisited for graph matching. PVLDB 3(1), 1161–1172 (2010), http://www.vldb.org/pvldb/vldb2010/pvldb_vol3/R103.pdf
11. Kaminski, M., Kostylev, E.V., Grau, B.C.: Semantics and expressive power of subqueries and aggregates in SPARQL 1.1. In: Proceedings of the 25th International Conference on World Wide Web, WWW 2016. pp. 227–238. ACM (2016)
12. Kiesel, N., Schürr, A., Westfechtel, B.: Gras, a graph-oriented (software) engineering database system. Inf. Syst. 20(1), 21–51 (1995), https://doi.org/10.1016/0306-4379(95)00002-L
13. Kostylev, E.V., Reutter, J.L., Ugarte, M.: CONSTRUCT queries in SPARQL. In: 18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium. pp. 212–229 (2015)
14. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34(3), 16:1–16:45 (2009), https://doi.org/10.1145/1567274.1567278
15. Robinson, I., Webber, J., Eifrem, E.: Graph Databases. O'Reilly Media, Inc. (2013)
16. SPARQL 1.1 Query Language. W3C Recommendation (march 2013), https://www.w3.org/TR/sparql11-query/
17. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation (February 2014), https://www.w3.org/TR/rdf11-concepts/
18. RDF Schema 1.1. W3C Recommendation (February 2014), www.w3.org/TR/2014/REC-rdf-schema-20140225/, www.w3.org/TR/2014/REC-rdf-schema-20140225/

A Proofs

First let us prove the results about colimits and POIM transformations in the category $\mathcal{G}_C(A)$. It can be helpful to remember that a morphism $a : T \to T'$ in $\mathcal{G}(A)$ and a map $M : |T| \to |T'|$ are such that $M(x) = |a|(x)$ for each attribute $x \in |T|$ if and only if $M^3(t) = a(t)$ for each triple $t \in T$.

Proposition 8. *Given graphs $T_1, ..., T_k$ on A such that $|T_i| \cap |T_j| \subseteq C$ for each $i \neq j$, the union $T_1 \cup ... \cup T_k$ is a coproduct of $T_1, ..., T_k$ in $\mathcal{G}_C(A)$.*

Proof. Consider morphisms $a_i : T_i \to T$ in $\mathcal{G}_C(A)$ for $i = 1, ..., k$ and the maps $|a_i| : |T_i| \to |T|$. Note that $|T_1 \cup ... \cup T_k| = |T_1| \cup ... \cup |T_k|$ and that $|T_1| \cup ... \cup |T_k|$ is the disjoint union of the sets $|T_i| \setminus C$ for $i = 1, ..., k$ and $(|T_1| \cup ... \cup |T_k|) \cap C$, because of the assumption $|T_i| \cap |T_j| \subseteq C$ for each $i \neq j$. Thus we can define a map $M : |T_1 \cup ... \cup T_k| \to |T|$ by: $M(x) = |a_i|(x)$ for each i and each $x \in |T_i| \setminus C$ and $M(x) = x$ for each $x \in (|T_1| \cup ... \cup |T_k|) \cap C$. Then M coincides with $|a_i|$ on $|T_i|$ for each i. Thus for each $t \in T_i$ we have $M^3(t) = a_i(t)$, which proves that the image of $T_1 \cup ... \cup T_k$ by M^3 is in T and that the restriction of M^3 defines a morphism $a : T_1 \cup ... \cup T_k \to T$ in $\mathcal{G}_C(A)$ which coincides with a_i on T_i for each i. Unicity is clear.

Proposition 9. *Let $l : L \to K$ and $m : L \to G$ be morphisms of graphs on A such that K is finite, l is an inclusion and m fixes C. Let us assume that $|G| \cap |K| \subseteq C$ (this is always possible up to isomorphism in $\mathcal{G}_C(A)$, by Remark 7). Let $N : |K| \to A$ be such that $N(x) = |m|(x)$ for $x \in |L|$ and $N(x) = x$ otherwise. Let $D = G \cup N^3(K)$, let $n : K \to D$ be the restriction of N^3 and $g : G \to D$ the inclusion. Then $|D| = |G| \cup |K \setminus L|$ and the square (l, m, n, g) is a pushout square in $\mathcal{G}_C(A)$.*

Proof. From $D = G \cup N^3(K)$ we get $|D| = |G| \cup |N^3(K)|$, and since $|N^3(K)| = N(|K|) = N(|L| \cup |K \setminus L|) = N(|L|) \cup N(|K \setminus L|) = |m|(|L|) \cup |K \setminus L|$ with $|m|(|L|) \subseteq |G|$ we get $|D| = |G| \cup |K \setminus L|$. The definition of n implies that $g \circ m = n \circ l$. Now let $a : G \to T$ and $b : K \to T$ be any morphisms in $\mathcal{G}_C(A)$ such that $a \circ m = b \circ l$. First, let us focus on attributes. We have $|g| \circ |m| = |n| \circ |l|$ and $|a| \circ |m| = |b| \circ |l|$. Since $|G| \cap |K \setminus L| \subseteq C$ we have $|a|(x) = |b|(x) = x$ for each $x \in |G| \cap |K \setminus L|$. Since $|D| = |G| \cup |K \setminus L|$ there is a unique map $F : |D| \to |T|$ such that $F(x) = |a|(x)$ for $x \in |G|$ and $F(x) = |b|(x)$ for $x \in |K \setminus L|$. Thus on the one hand $F(|g|(x)) = F(x) = |a|(x)$ for each $x \in |G|$, so that $F \circ |g| = |a|$. And on the other hand for each $x \in |K|$, if $x \in |L|$ then $F(|n|(x)) = F(|m|(x)) = |a|(|m|(x)) = |b|(|l|(x)) = |b|(x)$, otherwise $F(|n|(x)) = F(x) = |b|(x)$, so that $F \circ |n| = |b|$. Second, let us consider triples. Since $D = G \cup N^3(K)$ and $F^3(G) = a(G)$ and $F^3(N^3(K)) = F^3(n(K)) = b(K)$ we get $F^3(D) \subseteq T$, which means that there is a morphism $f : D \to T$ of graphs on A such that $|f| = F$, $f \circ g = a$ and $f \circ n = b$. Unicity is clear.

Proposition 14. *Let (L, R) be a basic construct query and $m : L \to G$ a match. Let $P : |R| \to A$ be defined by $P(x) = |m|(x)$ for $x \in |R|_V$ and $P(x) = x$ otherwise. Then, up to isomorphism in $\mathcal{Q}_I$, the result of applying $PoIm_{L,R}$ to m is $p : R \to H$ where $H = P^3(R)$ and p is the restriction of P^3.*

Proof. We use the notations of Diagram (1). Up to isomorphism in $\mathcal{Q}_I$ we can assume that all blanks in L or in R are distinct from the blanks in G. Then $|G| \cap |K| \subseteq C$, so that by Proposition 9 the data graph D is $D = G \cup n(K)$ where n is such that $|n|(x) = |m|(x)$ for $x \in |L|$ and $|n|(x) = x$ otherwise. It follows that the restriction of n to R is such that $|n|(x) = |m|(x)$ for $x \in |L| \cap |R|$ and $|n|(x) = x$ otherwise. Note that $|L| \cap |R|$ is the disjoint union of $|L|_I \cap |R|_I$, that is fixed by all morphisms in $\mathcal{Q}_I$, and $|L|_V \cap |R|_V$, with $|L|_V \cap |R|_V = |R|_V$ since $|R|_V \subseteq |L|_V$. Thus the restriction of n to R is such that $|n|(x) = |m|(x)$ for $x \in |R|_V$ and $|n|(x) = x$ otherwise. The result follows.

Now let us consider the basic construct queries. The semantics of SPARQL CONSTRUCT queries is defined in [13, Section 5], based on the seminal paper [14]. In order to express this definition we have to introduce some terminology and notations. Note that in [13] literals are allowed as subjects or predicates in RDF graphs. However for our purpose this does not matter, so that we stick to the "official" definition of an RDF graph from [17]. Note that for each subset T of $(IBV)^3$ and each subset X of $|T|$, each map $f : X \to IBV$ gives rise to a map $f' : |T| \to IBV$ such that $f'(x) = f(x)$ when $x \in X$ and $f'(x) = x$ otherwise, then $f' : |T| \to IBV$ gives rise to $f'' : T \to (IBV)^3$ which is the restriction of $(f')^3$ to T. There will not be any ambiguity in denoting f not only the given f but also its extensions f' and f'', so that we can state the following definitions from [13]. For simplicity we consider only the SPARQL queries "CONSTRUCT $\{R\}$ WHERE $\{L\}$" such that each variable in R occurs in L. Indeed, variables outside $|L|_V$ cannot be instanciated in the result, and according to [16, Section 16.2], if a triple contains an unbound variable, then that triple is not included in the output RDF graph. Thus, triples involving a variable in $|R|_V \setminus |L|_V$, if any, can be dropped. It is assumed in [13] that there is no blank in L. Indeed, since blank nodes in graph patterns act as variables, each blank in L can be replaced by a new variable. A *solution mapping* (or simply a *mapping*) from a basic graph pattern L to an RDF graph G is a map $\mu : |L|_V \to IB$ such that $\mu(L) \subseteq G$. When L and R are basic graph patterns such that $|R|_V \subseteq |L|_V$, the *answer* of the SPARQL query "CONSTRUCT $\{R\}$ WHERE $\{L\}$" over an RDF graph G is the set of all well-formed triples $\mu(f_\mu(t))$ for all triples $t \in R$ and all mappings μ from L to G, where for each μ a map $f_\mu : |R|_B \to B$ is chosen in such a way that the subsets $f_\mu(|R|_B)$ of B are pairwise distinct and all of them are distinct from $|G|_B$.

Theorem 19. *Let L and R be basic graph patterns with $|L|_B = \emptyset$ and $|R|_V \subseteq |L|_V$. Then (L, R) is a basic construct query and the set of well-formed triples in the query result of applying (L, R) to an RDF graph G is isomorphic in $\mathcal{D}_I$ to the answer of the SPARQL query "CONSTRUCT $\{R\}$ WHERE $\{L\}$" over G.*

Proof. Clearly (L, R) is a basic construct query and $|G|_B \cap |L|_B = \emptyset$. We can assume without loss of generality that $|G|_B \cap |R|_B = \emptyset$. The query result H of applying (L, R) to G is given by Definition 18, as reminded now. Let m_i $(i = 1, ..., k)$ be the matches from L to G and for each i let H_i be the data graph obtained from R by replacing each variable x in R by $m_i(x)$ and each blank

in R by a new blank, then $H = H_1 \cup ... \cup H_k$. The Theorem now follows from the remark that the maps μ'' on triples which are associated (as above) to the mappings μ are precisely the matches from L to G.

Proposition 23. *Let (L, R) be a basic construct query and G a data graph. The high-level query result of (L, R) against G is isomorphic, in the category $\mathcal{D}_I$, to the query result of (L, R) against G.*

Proof. This is a consequence of the description of the result of a POIM transformation from Proposition 14.

Proposition 27. *Let (L, R) be a basic construct query and G a data graph. The low-level query result of (L, R) against G is isomorphic, in the category $\mathcal{D}_I$, to the query result of (L, R) against G.*

Proof. This is a consequence of the description of the result of a POIM transformation from Proposition 14 and the description of coproducts in $\mathcal{D}_I$ from Proposition 8.

Finally let us look at the basic select queries. For the semantics of SPARQL SELECT queries we rely on [11, Section 2].

Proposition 30. *Each relational data graph $S = \{(\_ : l_i, s_j, y_{i,j})\}_{i\in\{1,...,k\},j\in\{1,...,n\}}$ determines a multirelation $Rel(S) = \{(y_{i,1}, ..., y_{i,n})\}_{i\in\{1,...,k\}}$ of arity n on IB.*

Proof. This result is clear from the definitions of relational data graphs and multirelations.

Proposition 35. *Let (L, S) be a basic select query and G a data graph. The query result of $(L, Gr(S))$ against G is a relational data graph H. More precisely, let $S = \{?s_1, ..., ?s_n\}$ and let $m_1, ..., m_k$ be the matches from L to G, then H is the set of triples $(\_ : l_i, s_j, m_i(?s_j))$ where $i \in \{1, ..., k\}$, $j \in \{1, ..., n\}$, and the blanks $\_ : l_1, ..., \_ : l_k$ are pairwise distinct.*

Proof. We have $Gr(S) = \{(\_ : r, s_j, ?s_j)\}_{j\in\{1,...,n\}}$, so that according to Definition 18 the query result of $(L, Gr(S))$ against G is $H_1 \cup ... \cup H_k$ where $H_i = \{(\_ : l_i, s_j, m_i(?s_j))\}_{j\in\{1,...,n\}}$ and the blanks $\_ : l_1, ..., \_ : l_k$ are pairwise distinct.

Theorem 37. *Let L be a basic graph pattern of SPARQL with $|L|_B = \emptyset$ and $S = \{?s_1, ..., ?s_n\}$ a finite set of variables included in $|L|_V$ and let G be an RDF graph. Then the query result of (L, R) against G is the answer of the SPARQL query "SELECT $?s_1, ..., ?s_n$ WHERE $\{L\}$" over G.*

Proof. According to [11, Section 2], the answer of the SPARQL SELECT query is the multiset with elements the restrictions $\mu|_S$ of the mappings μ from L to G to the subset S of $|L|_V$, each $\mu|_S$ with multiplicity the number of corresponding μ's. Since the mappings from L to G correspond bijectively to the matches from L to G, the result follows from Proposition 35.